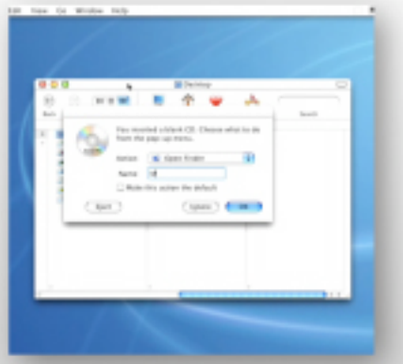
CHUD—Performance Optimization Tools in Depth

Session 506



Optimizing for G5

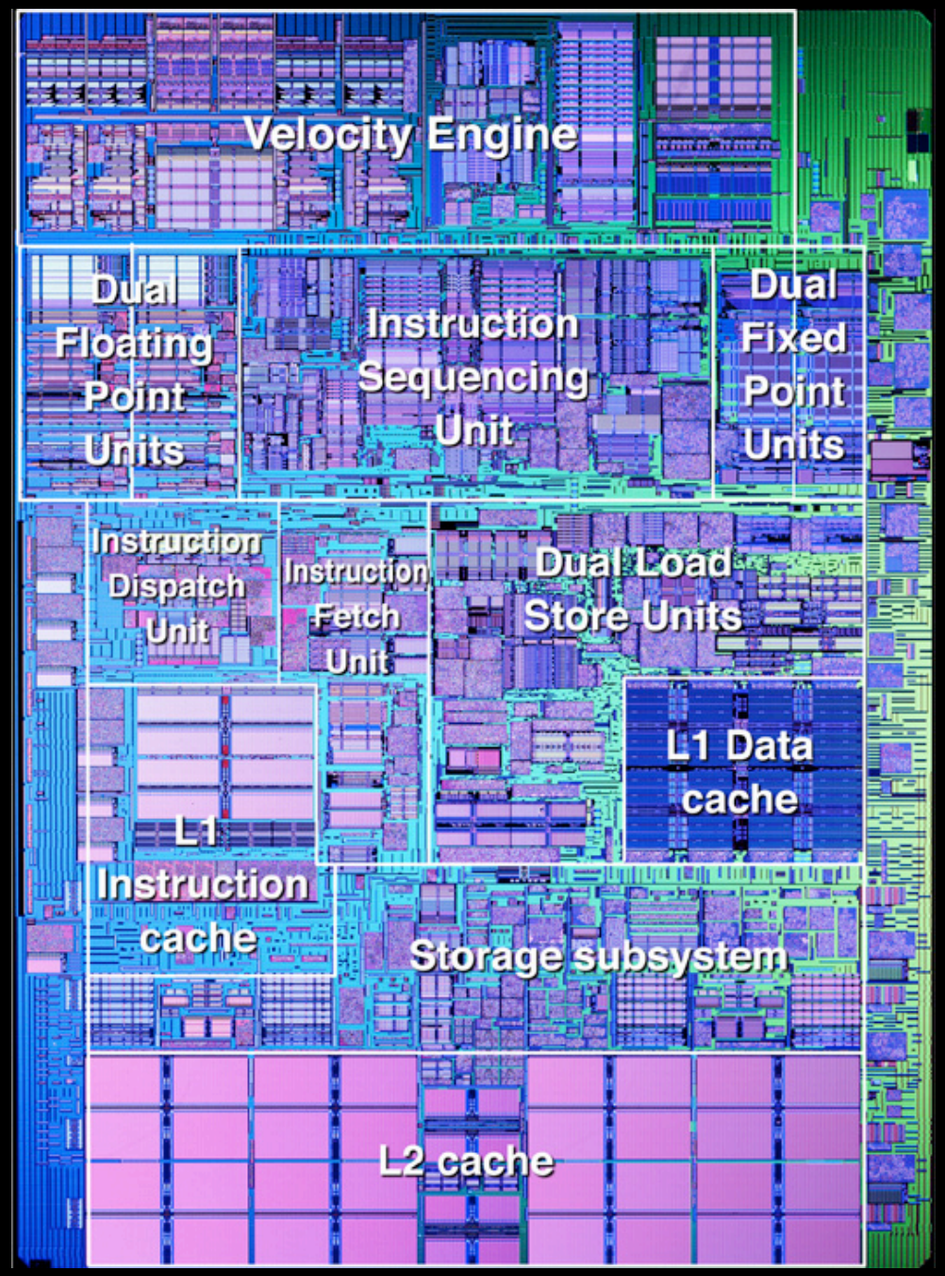
Sanjay Patel
Architecture and Performance Group
chud-tools-feedback@group.apple.com

PowerPC 970

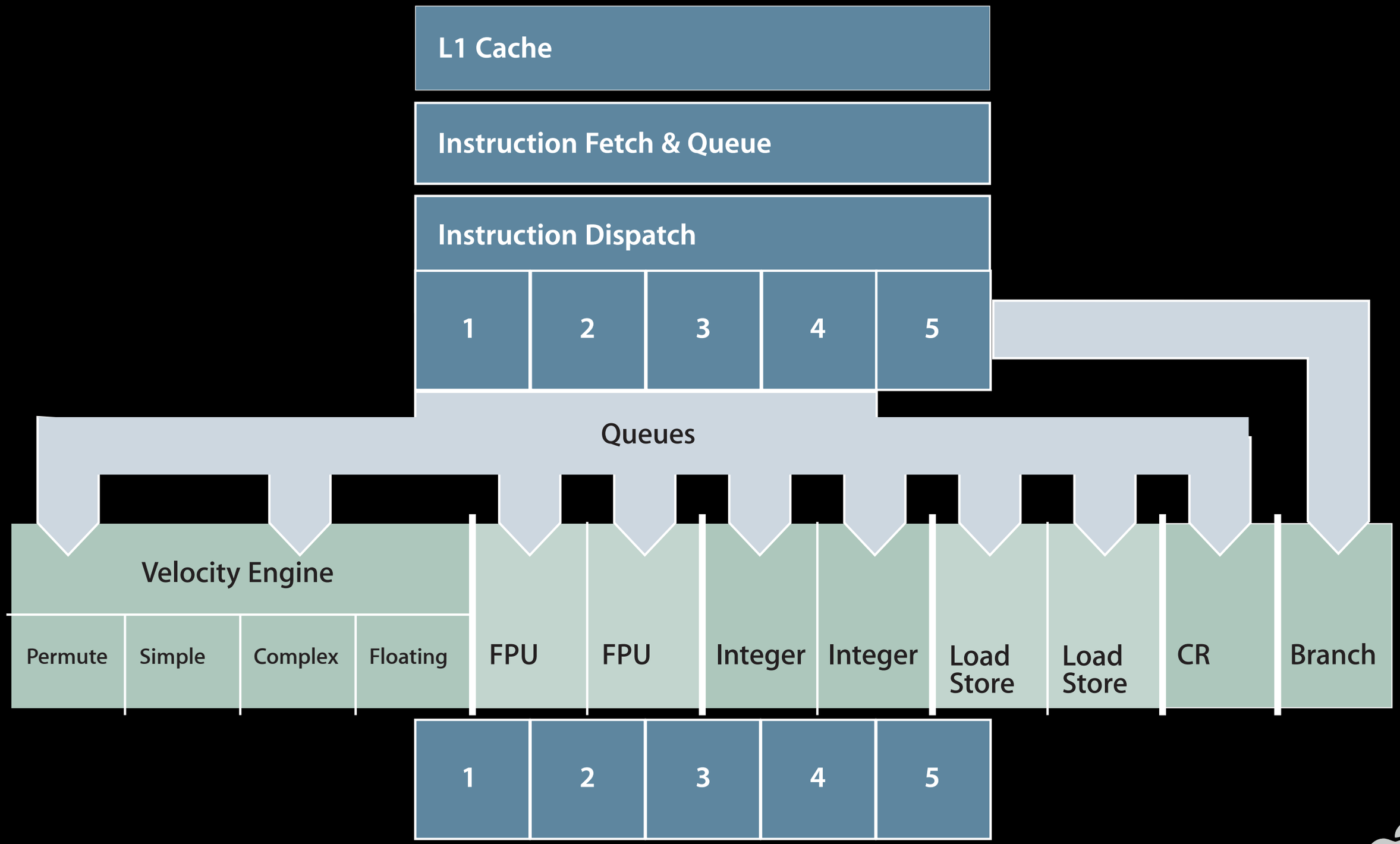
- High-performance processing
  - 64-bit PowerPC™
  - Based on IBM POWER4™
  - Advanced superscalar design
  - Super-pipelined micro-architecture
- Accelerated computation using AltiVec™
- High bandwidth point-to-point processor interface
- Automated hardware prefetch engines

PPC970 Processing Power

- Two load/store units
- Two fixed point units
- Two IEEE floating point units
- Four vector (AltiVec™) sub-units
- Branch unit
- Condition register unit



Massively Parallel Execution Core



PPC970 vs. PPC7450

	PPC970	PPC7450
Instructions in Core	200+	30+
Pipeline Stages	16	7
Load/Store	2 LSUs	1 LSUs
Floating Point	2 FPUs	1 FPU
Integer	2 FXUs	3 SFXU, 1 CFXU
Vector	VALU, VPERM	VALU, VPERM

PPC970 vs. PPC7450

	PPC970	PPC7450
Cache Line Size	128 B	32 B
L1 Data Cache	32 KB	32 KB
	2-Way Associative	8-Way Associative
	Write Through	Write Back
L1 Inst Cache	64 KB	32 KB
	Direct Mapped	8-Way Associative

PPC970 vs. PPC7450

	PPC970	PPC7450
L2 Cache	512 KB	256 KB
	8-Way Associative	8-Way Associative
	LRU	Random
L3 Cache	None	0 to 2 MB
Processor BW	3.5 GB/s Each Way	1.3 GB/s Combined
Memory BW	6.4 GB/s	2.7 GB/s

Performance Solutions for PPC970

Instruction latency

- Problem
  - Most instructions have longer latencies than PPC7450
- Solution
  - Do more in parallel
    - Manually unroll important loops or use `-funroll-loops` with GCC
    - Schedule code by using `-mtune=970` with GCC 3.3

Performance Solutions for PPC970

Branch mispredictions

- Problem
  - Longer pipelines make branch mispredictions more expensive
- Solution
  - Use `__builtin_expect(expr,value)` with GCC or `++/--` in assembly for highly predictable branches
  - Use branch-less comparisons
    - `FSEL` (`-ffast-math`), `VSEL`, carry bit for integer comparisons
    - Use feedback-directed optimization (`-fbranch-probabilities`) with GCC

Performance Solutions for PPC970

Data cache

- Problem
  - L1 data cache is different than on PPC7450
    - 128-byte cachelines, 32KB 2-way associative
    - Write-through, does not allocate on store misses
- Solution
  - Design algorithms and data structures to maximize locality
  - Contiguous memory accesses trigger HW prefetcher

Performance Solutions for PPC970

Memory latency

- Problem
  - Latency to main memory higher than PPC7450 systems
- Solution
  - Use SW and HW prefetch to reduce effective latency
  - “Batch” loads to take advantage of high BW

Performance Solutions for PPC970

Data stream touch instruction

- Problem
  - DST is execution serializing
- Solution
  - Remove DST instructions from code
  - Rely on HW prefetcher for contiguous memory accesses
  - Replace with DCBT

Performance Solutions for PPC970

Cache control instructions

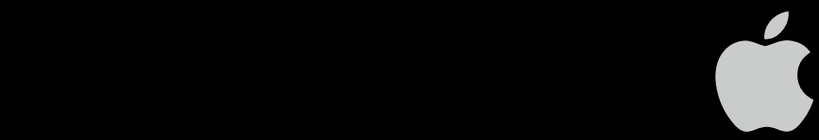
- Problem
  - Legacy code that uses `DCBZ` or `DCBA` will perform poorly
- Solution
  - Remove `DCBZ` and `DCBA` instructions from code
  - Use `DCBZL` if needed

How to Use DCBZL

```
#define __dcbzl(base, index)     \
  __asm__ ("dcbzl %0, %1" : /*no result*/ : \
           "b%" (index), "r" (base) : "memory")

// ****** WARNING: JUST USE MEMSET IF YOU CAN! ******

void fastOp(void *base, uint32_t len, uint32_t lineSize)
{
    for(uint32_t offset=0; offset<len; offset+=lineSize) {
        __dcbzl(base, offset);
        // do what memset() doesn't do here
    }
}
```



Performance Solutions for PPC970

Synchronization primitives

- Problem
  - Lock and serialization primitives are costly
- Solution
  - Minimize lock hold times and avoid locks if possible
  - Ensure locks are spaced at least one cache line apart

Performance Solutions for PPC970

Instruction scheduling

- Problem
  - Recompiling or hand scheduling recommended for optimal performance
- Solution
  - Schedule code by using `-mtune=970` with GCC 3.3
  - Understand dispatch group formation (use Shark...)

Performance Solutions for PPC970

Summary

- Unroll and schedule code
  - Two pipelined double-precision FPUs
  - Two independent LSUs
  - 80 renames for each register file
  - 200+ instructions in flight
- Use AltiVec™ to calculate up to 32 GFLOPS
- Use full-precision hardware square root

  `-mpowerpc-gpopt`
- Native `long long` support for 64-bit integer math in leaf functions

  `-mpowerpc64`

Performance Solutions for PPC970

Summary

- Take advantage of high memory bandwidth
  - 32 GB/s to L1 data cache
  - 64 GB/s between L1-L2 caches
  - 3.5 GB/s to and from main memory
- Rely on hardware prefetch engines to stream data
- Use optimal cache control instructions
  - `DCBT`—lower cost than `DST`
  - `DCBZ`—emulated to affect 32-bytes; use `DCBZL` instead
  - `DCBA` and `DCBI` are illegal instructions on PPC970

CHUD Tools
Computer Hardware Understanding Developer Tools

Nathan Slingerland
Architecture and Performance Group
chud-tools-feedback@group.apple.com

Introduction to CHUD Tools

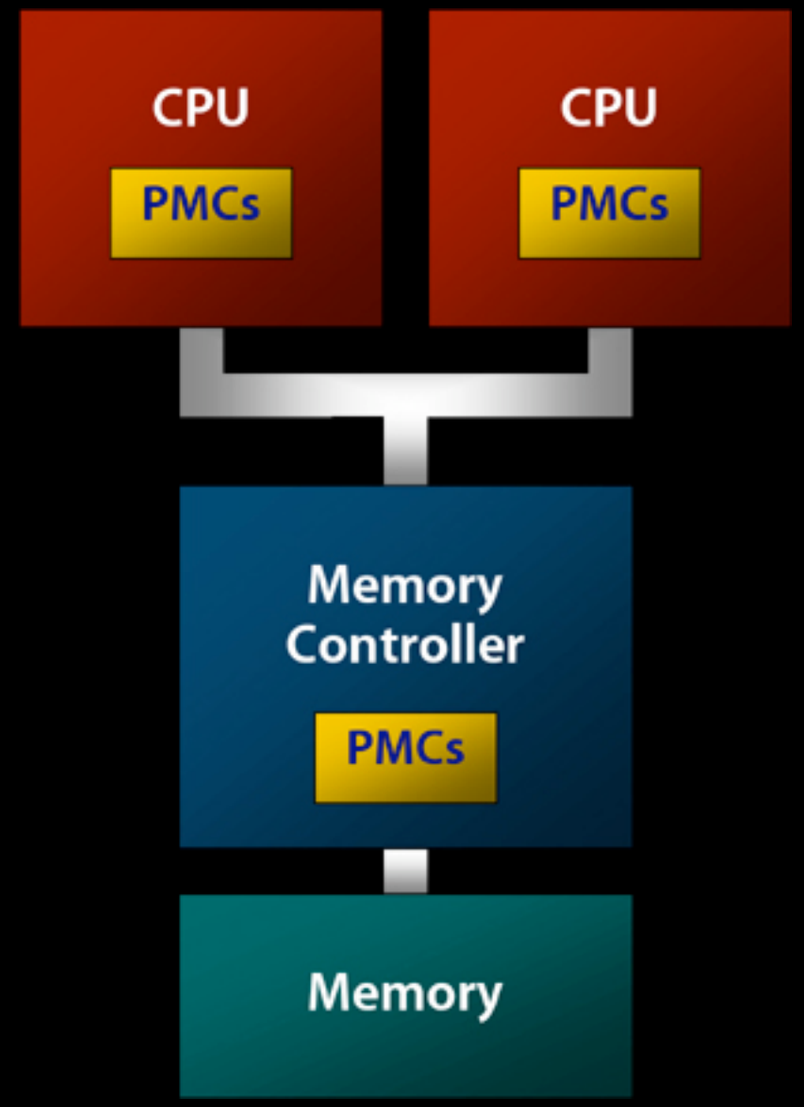
- Suite of performance analysis tools
- Low-level access to performance monitors
  - Processor
  - Memory controller
  - Operating system
- Helps you find problems and improve your code
- Freely available on the web and the developer tools CD!

CHUD Tools 3.0

- Profiling
  - Shark—instruction-level profiler
  - MONster—spreadsheet for performance events
  - Saturn—function call graph visualizer
- Tracing
  - Amber—instruction tracer
  - Acid—trace analyzer
  - SimG4 and SimG5—cycle-accurate CPU simulators
- Instrumentation
  - CHUD.framework—API for performance tuning

Performance Counters

- Apple systems have performance monitor counters (PMCs)
- PMCs record performance “events”
  - Cache misses
  - Instructions completed
  - Execution stall cycles
  - Page faults
- Use CHUD Tools to configure, control and view the PMCs



Shark

- Profile a process, thread or entire system
  - Time profile—visualize performance hot spots
  - Event profile—relate hardware events to source
  - Capture everything—drivers, kernel, applications
  - Very low overhead
- Automated Analysis
  - Annotated source code and disassembly
  - Static analysis to find sub-optimal code
  - Optimization tips
- Command line version
- Save and review sessions

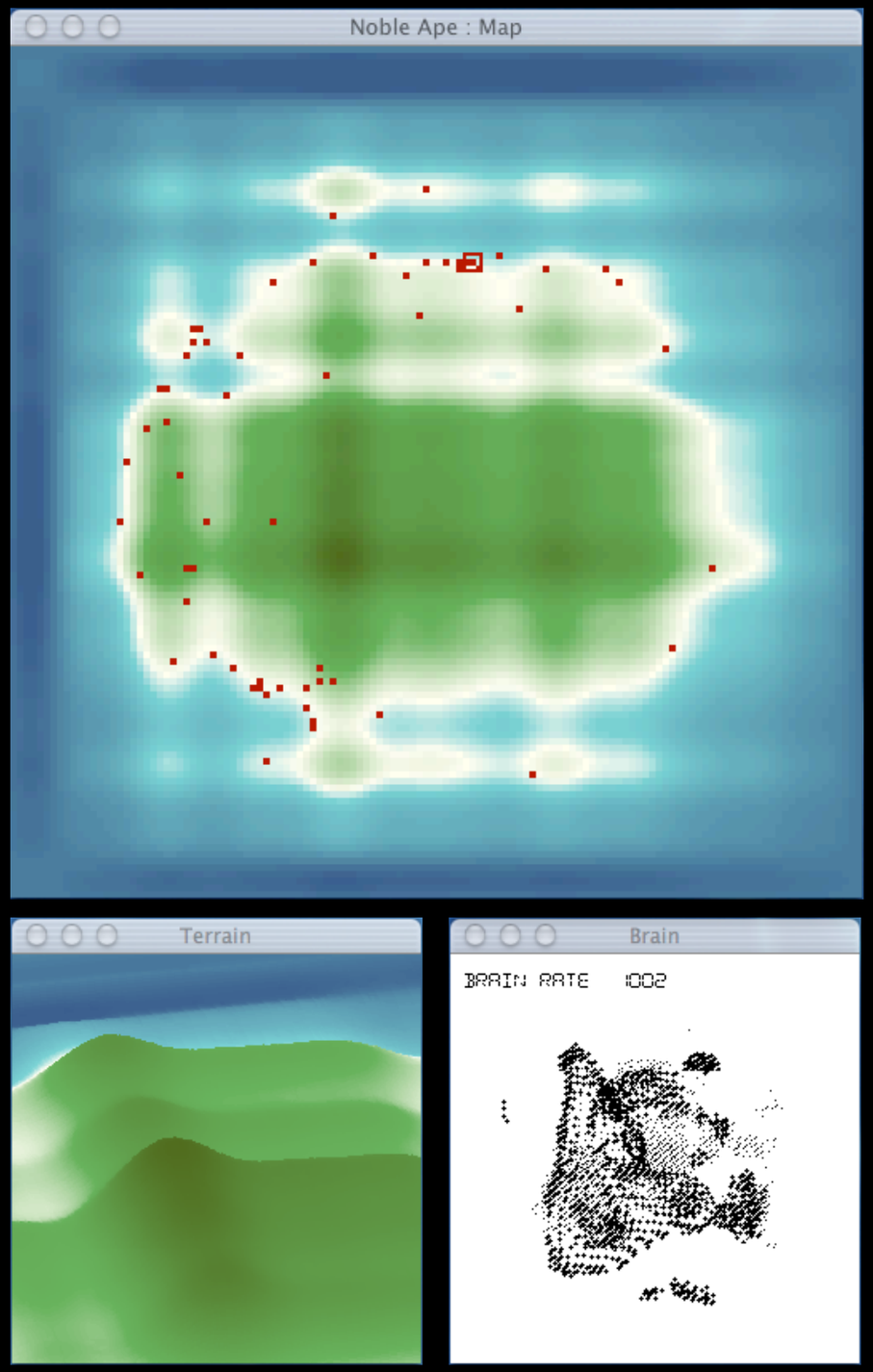
MONster

- Configure Performance Monitor Counters
- Collect PMC data based on:
  - Timed intervals, event counts or hotkey
- View results in spreadsheet or chart form
- Shortcuts—compute performance metrics:
  - Memory bandwidth, cycles per instruction
  - Define your own with infix expressions
- Command line version
- Save and review sessions

Noble Ape Simulation

Demo

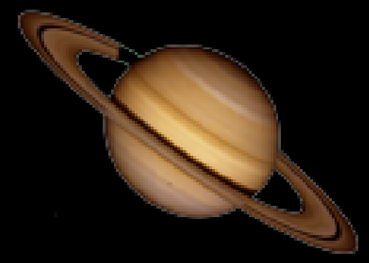
- Tom Barbalet's simulation of apes on a tropical island
  - Biological environment
  - Cognitive processes
- Open source
- For more information:
  - http://www.nobleape.com/sim/



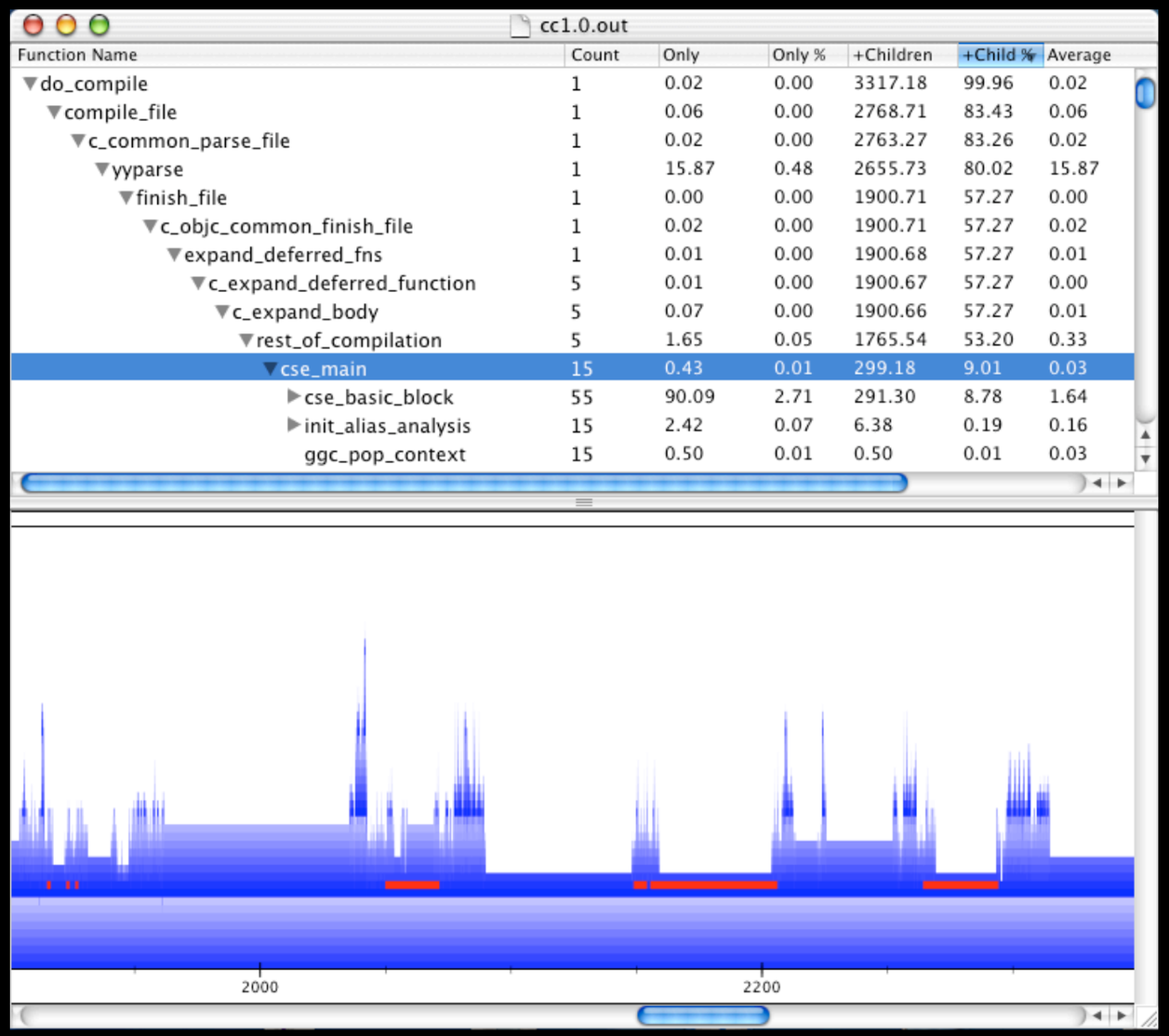
Noble Ape Performance Summary

	G4 - Dual 1.42GHz	G5 - Dual 2GHz
Scalar	1200	1330
Threaded	2080	2490
Vector	4510	10200
Vector Optimized	5700	16140

Saturn



- Exact profile
- Visualize call tree
- Instruments functions at entry/exit with GCC
- Record function call history
- For each function:
  - Call counts
  - PMC event counts
  - Execution times



CHUD.framework

- Instrument your source code
  - Start and stop MONster or Shark
- Write your own performance analysis tools
  - Setup, start and stop PMCs
  - Collect information about system hardware
    - CPU, memory controller, bus
- Extensive HTML reference guide

CHUD.framework Example #1

```
#include <CHUD/chud.h>
…
chudInitialize();

// MONster or Shark is in remote mode
chudAcquireRemoteAccess();

chudStartRemotePerfMonitor(“run_01”);

your_important_function();

chudStopRemotePerfMonitor();

chudReleaseRemoteAccess();
```

CHUD.framework Example #2

```
chudInitialize();
chudAcquireSamplingFacility();

chudSetCounterEvent(chudAllCPUDev, PMC_1, 1);
chudSetCounterEvent(chudAllCPUDev, PMC_2, 2);

chudClearCounters();
chudStartCounters();

important_function();

chudStopCounters();

do_something_with_results();
chudReleaseSamplingFacility();
```

Instruction Tracing and Simulation

- Amber—instruction tracer
- Acid—trace analyzer
  - Trace statistics
  - Memory footprint (gnuplot output)
  - Points out problematic instruction sequences
  - Dependency stalls, spilled loads, unaligned operands
- SimG4—PPC7400 cycle-accurate simulator
- SimG5—PPC970 cycle-accurate simulator

Roadmap

305 Tuning Software with Performance Tools	The Presidio Wed., 5:00pm
507 Mac OS X High Performance Libraries	The Marina Thu., 9:00am

Who to Contact

Mark Tozer-Vilchez
Desktop Hardware Evangelist
tozer@apple.com

CHUD Feedback
chud-tools-feedback@group.apple.com

http://developer.apple.com/wwdc2003/urls.html

For More Information

On the Web:
http://developer.apple.com/tools/performance/

AltiVec™ Web Page
http://developer.apple.com/hardware/ve/

Noble Ape
http://www.nobleape.com/sim/

Q&A

Mark Tozer-Vilchez
Desktop Hardware Evangelist
tozer@apple.com

http://developer.apple.com/wwdc2003/urls.html

WWDC 2003

Worldwide Developers Conference